Management with Condor, Part 1

By Forrest Hoffman

A good job queuing and scheduling system is required whenever more than a couple of researchers share a Beowulf cluster. Coordinating with other users about when and where to run jobs on a shared cluster isn't impossible, but cluster administrators quickly realize the importance of having a robust batch system once users begin competing for resources.

One popular batch system, *OpenPBS* (Portable Batch System), was discussed in the October 2002 issue of this column (http://www.linux-mag.com/2002-10/extreme_01.html), and the *Maui* scheduler was covered in November 2002 (http://www.linux-mag.com/2002-11/extreme_01.html). OpenPBS consists of a job server, a job executor, and a job scheduler. Maui is an advanced batch scheduler that may be used in place of the default job scheduler provided in OpenPBS. Maui decides where, when, and how to run jobs, based on specified policies, priorities, and resource limitations.

Another batch system, named *Condor* (http://www.cs.wisc.edu/condor), is increasingly being used in research environments for managing compute-intensive jobs on both Beowulf clusters and disparate collections of desktops and workstations. Like OpenPBS and other batch systems, Condor provides a queuing mechanism, scheduling policy, job priority scheme, and resource classification.

However, unlike most other batch systems, Condor doesn't require dedicated compute servers. It can harness otherwise idle machines by checkpointing and migrating jobs to those computers (when migrated and restarted, the job continues precisely where it left off). In addition, Condor can order job execution as specified by the user, and it enables grid computing by executing jobs on participating computers or clusters in various locations worldwide.

Condor is developed by the Condor Team at the University of Wisconsin, Madison, where it's been used and developed for more than ten years. It's recently become an Open Source project, but the source code has not yet appeared on Wisconsin's web site. Condor runs under HP-UX, Solaris, IRIX, Digital Unix, Tru64, Mac OS X, and, of course, Linux. Condor is particularly well-suited to running the same job hundreds of times with different input data sets or parameters.

Condor uses *ClassAds* — analogous to classified advertisements in the newspaper — to match job requirements specified by the user (called *job ClassAds*) with advertised resource attributes (called *machine ClassAds*), like available memory, CPU type and speed, and current load average. The machine ClassAd also advertises the conditions under which it is willing to run a job, and what type of job it prefers. While the policy attributes for a machine in a Beowulf cluster might be permissive, policy attributes for a co-worker's workstation might specify that it's willing to run jobs only at night and when there's no keyboard activity on the machine.

Condor Universes

A *universe* in Condor defines the environment in which a job is executed. Supported universes are `standard`, `vanilla`, `pvm`, `mpi`, `globus`, `java`, and `scheduler`. The `universe` attribute for a job is specified in the *submit description file*, a small text file submitted with the job that describes its requirements, the name of the executable, and the names of any input, output, and log files.

The `standard` universe provides process migration and remote system calls, but has some restrictions about what programs can do. The `vanilla` universe provides fewer services, but has very few restrictions.

The `pvm` and `mpi` universes provide support for programs written in PVM (Parallel Virtual Machine) and MPI (Message Passing Interface, specifically MPICH), respectively.

The `globus` universe allows users to submit Globus (http://www.globus.org) jobs through Condor, and the `java` universe supports jobs written for the Java Virtual Machine (JVM).

The `scheduler` universe allows a job to be executed immediately, without preemption, on the machine where the job is submitted; the job doesn't wait to be matched with a machine.

While running a job in the `standard` universe, Condor checkpoints the program at regular intervals creating a *checkpoint image* (a snapshot of the current state of the program and its memory). This image may be used to restart the job on a new machine if the machine currently running the job should crash or fail. A checkpoint image is also generated whenever Condor decides to migrate a job from one machine to another; the image is copied to the new machine, restarting from where it left off on the previous machine.

Remote systems calls in the `standard` universe make a job think it's executing on the machine on which it was started. When a job runs on a remote machine, a second process called a *condor_shadow* runs on the submitting machine. When the remote job attempts to make a system call or perform file I/O, the *condor_shadow* process performs the system call or file I/O instead, and returns the results to the remote machine.

To prepare a program to run in `standard`, it must be relinked (but not recompiled) using *condor_compile*. The Condor libraries (linked to the program) intercept system calls and perform checkpointing while the program is running.

To use the `standard` universe, a program must conform to a number of restrictions. No multi-process jobs are allowed (namely, avoid `fork()`, `exec()`, `system()`, etc.); interprocess communication is not allowed; network communication must be brief; the use of signals `SIGUSR2` and `SIGTSTP` is prohibited; and alarms, timers, and sleeping are not allowed. Additional restrictions are described in the documentation.

The `vanilla` universe is for programs that can not be successfully re-linked with the Condor libraries. Jobs run in this universe do not checkpoint or use remote system calls. When a machine running such a job becomes unavailable, Condor can either suspend the job (in hopes of completing it at a later time), or it can restart the job from the beginning on another machine in the available pool. Under Unix, Condor assumes that a shared file system is available for `vanilla` jobs, since I/O can't be forwarded to the submitting host. Alternatively, a file transfer mechanism in Condor may be used to move files to the executing host.

Condor Pools and Daemons

A Condor *pool* consists of a single machine, the *Central Manager,* and a number of other machines that join the pool as participating resources. The Central Manager periodically receives status updates from the pool, and tries to match pending requests with appropriate, available resources. The *condor_collector* daemon runs on the Central Manager, and receives ClassAds describing the state of all other daemons in the pool. The *condor_negotiator* daemon performs all matchmaking between jobs and resources: it queries the scheduler daemons on each machine for resource requests, and is responsible for enforcing user priorities.

Any machine in a pool can be configured to execute Condor jobs, including the Central Manager. Each execution host runs the *condor_startd* daemon, which advertises the machine's resource attributes. The *condor_starter* program actually spawns remote Condor jobs on a given machine: it establishes the runtime environment and monitors the running job.

Any machine in a pool can be configured to allow or disallow job submissions. Submit hosts require a fair amount of resources, since every job submitted from that machine has a corresponding *condor_shadow* process handling system calls and file I/O. The *condor_schedd* daemon, which also runs on submit hosts, represents resource requests to the Condor pool. Once a job's been matched to a resource, *condor_schedd* spawns the *condor_shadow* needed to serve that job.

A single machine in the pool may optionally be configured as a *Checkpoint Server* (running the *condor_ckpt_server*). This machine stores all checkpoint files for submitted jobs in the *standard* universe, so it requires lots of disk space and good connectivity to the rest of the pool.

The *condor_master* daemon is the top-level Condor daemon. It keeps all other daemons running on machines in the pool. *condor_master* runs on each machine in the pool regardless of the functions the machine performs.

Installing Condor on a Cluster

The process of installing Condor varies, depending on the type of pool being established. For a cluster with dedicated nodes, some basic assumptions are usually made: the front-end node is the Central Manager, the pool members share a filesystem (usually */home*), and all pool members are execution hosts.

The Condor distribution can be downloaded from the Condor download page at http://www.cs.wisc.edu/condor/downloads. The following example installation was performed with Condor version 6.5.3 under Red Hat Linux 7.3. Before beginning the installation, make sure that the fully qualified domain name of the front-end node is specified in */etc/hosts*.

While not absolutely necessary, creating a *condor* user on every node in the cluster simplifies installation. This is usually accomplished, as shown in *Figure One*, by running *adduser* followed by a script that propagates the password, shadow, and group files to the other nodes. Here *ssync* serves that purpose. Since this creates a home directory for the *condor* user on every node, that's where the software should be installed.

Now that those two details are out of the way, the Condor distribution should be downloaded and unpacked in some location like */usr/local/src/*. After moving to the resulting *condor-6.5.3/* directory, execute the Perl script `condor_install` on the front-end node to launch the installation process.

If you'd like to follow along closely with the next few paragraphs, you can download a transcript of a complete install from a cluster with a shared filesystem from http://www.linuxmagazine.com/downloads/2003-10/extreme/transcript1.txt.

The first time *condor_install* is run, select a full installation in step 1. All the nodes are listed in step 2, and you should force the install to */home/condor* in step 3, so that it's acces-

FIGURE ONE: Starting the installation procedure

```
[root@node001 root]# adduser condor
[root@node001 root]# ssync
[root@node001 root]# cd /usr/local/src
[root@node001 src]# tar xvzf
  condor-6.5.3-linux-x86-glibc22.tar.gz
[root@node001 src]# cd condor-6.5.3
[root@node001 condor-6.5.3]# ./condor_install
Welcome to condor_install.  You are going to
  need to answer a few
questions about how you want Condor configured
  on this machine, what
```

sible to all nodes in the cluster. The system administrator's email address is set in step 4. Since most clusters have a single password file that's copied to each node, choose that as the convention in step 5, as shown in *Figure Two*.

If you want to use the `java` universe, the path for your JVM should be entered in step 6.

The installer can create soft links for binaries and scripts that users need to use Condor in step 7; however, for this installation we simply add */home/condor/bin* to users' paths instead. The name of the Central Manager is provided in step 8. In steps 9 and 10, tell the installer to create configuration files for all nodes on the shared filesystem, and set the pool name. A soft link to the Condor config file is created in step 11.

Once the initial installation is complete, *condor_init* should run on all the nodes to create the local lock file directory on each node at */var/lock/condor* like in *Figure Three*.

FIGURE THREE: Establishing lock files

```
[root@node001]# brsh /home/condor/sbin/condor_init

***** node001 *****
/home/condor/condor_config already exists.
/home/condor/hosts/node001/log already exists.
/home/condor/hosts/node001/spool already exists.
/home/condor/hosts/node001/execute already exists.
/home/condor/hosts/node001/condor_config.local already exists.
Condor has been initialized, but not started.

***** node002 *****
/home/condor/condor_config already exists.
/home/condor/hosts/node002/log already exists.
/home/condor/hosts/node002/spool already exists.
/home/condor/hosts/node002/execute already exists.
Creating /home/condor/hosts/node002/condor_config.local

Creating /var/lock/condor
Condor has been initialized, but not started.
```

Next, and although it's not clear from the documentation, *condor_install* must be run a *second time* on the front-end node to establish it as the Central Manager. In the process, some information must be repeated even though it was provided in the first invocation of *condor_install*.

The online file http://www.linuxmagazine.com/downloads/2003-10/extreme/transcript2.txt details the second invocation of *condor_install*. Be sure in step 1 to setup this host as a Condor Central Manager. In step 3, specify the pool name again.

After the script completes, the *condor_master* daemon can be started on every cluster node. The status of all nodes can be checked by running *condor_status* from the binary directory at */home/condor/bin* as shown in *Figure Four* (pg. 62). If all nodes report in, everything should be fine.

To complete the installation, files in */etc/profile.d* should be created or edited to add */home/condor/bin* to users' default paths. In addition, the Condor startup file should be copied */etc/rc.d/init.d* and configured to run with *chkconfig*, so that the Condor daemons are started automatically when each node boots.

See Extreme Linux, *pg. 62*

FIGURE TWO: Steps to configure a cluster with a shared password file

```
To correctly run all jobs in your pool, including ones that aren't relinked for Condor, you must tell Condor if
  you have a shared filesystem, and if so, what machines share it.

Please read the "Configuring Condor" section of the Administrator's manual (in particular, the section "Shared
  Filesystem Config File Entries") for a complete explanation of these (and other, related) settings.

Do all of the machines in your pool from your domain ("cluster.ornl.gov") share a common filesystem? [no] yes

Configuring all machines to use "cluster.ornl.gov" for their filesystem domain.

Do all of the users across all the machines in your domain have a unique UID (in other words, do they all share a
  common passwd file)? [no] yes

Configuring all machines to use "cluster.ornl.gov" for their uid domain.

In some cases, even if you have unique UIDs, you might not have all users listed in the password file on each machine.
Is this the case at your site? [no]

Press enter to continue.
```

Guru Guidance, *from pg. 40*

insert a disk, click the icon to reactivate it, and then try accessing it again. Sometimes you can omit some of these steps, though.

➤ **MOUSE.** When you activate the mouse, Bochs takes over control of the mouse, which can make it hard to switch between other programs. Clicking the middle button toggles the mouse on and off, enabling you to work with other programs along with Bochs.

➤ **SNAPSHOTS.** Click the Snapshot icon to create a dump of the contents of a text display to a text file (*snapshot.txt*). If you want to create a screen shot of a graphics mode, you can use Linux graphics utilities to capture a screen shot of Bochs, including its icon bar and window widgets. If desired, you can then trim the image to the emulated OS alone. The GIMP is an excellent, if big, program that can handle this job.

➤ **CONFIGURATION.** Clicking the Config icon brings up a configuration tool (typically in the console in which Bochs is running). You can adjust various options normally set in the configuration file, tweaking them as necessary for the software you're running.

Bochs isn't without its limitations, of course. As already noted, it's quite slow compared to running programs directly, although, of course, running x86 programs directly isn't possible if you're using a non-x86 host computer.

Bochs is also limited in its video mode: by default, it provides VGA (640x480) resolution at best, which can be quite limiting.

Nonetheless, for some purposes Bochs can be extremely useful. You can run multiple operating systems on one computer, test a new OS or OS version without disturbing your working configuration, run x86 operating systems on non-x86 hardware, emulate networking, and so on. The Bochs web site offers a great deal of documentation, including many tips and techniques for common tasks, like making disk images and running multiple operating systems. Need emulation? Think inside the Bochs.

Roderick W. Smith is the author or co-author of eleven books, including Advanced Linux Networking *and* Linux Power Tools. *He can be reached at rodsmith@rodsbooks.com.*

Extreme Linux, *from pg. 44*

More to Come!

Now that Condor is installed on the cluster, it's ready to receive job requests. Next month, we'll learn more about Condor's policy and priority schemes, and see how to submit and manage MPI and serial jobs in different Condor universes to optimize resource utilization in a Beowulf cluster environment.

Stay tuned!

Forrest Hoffman is a computer modeling and simulation researcher at Oak Ridge National Laboratory. He can be reached at forrest @climate.ornl.gov. You can view the transcripts for the Condor installations used in this column at http:// www.linuxmagazine.com/ downloads/2003-10/extreme.

FIGURE FOUR: Checking the status of Condor nodes

```
[root@node001]# /home/condor/bin/condor_status

Name           OpSys       Arch   State      Activity   LoadAv Mem
ActvtyTime

vm1@node001.c  LINUX       INTEL  Unclaimed  Idle        0.000  1008
0+00:00:08
vm2@node001.c  LINUX       INTEL  Unclaimed  Idle        0.000  1008
0+00:00:05
vm1@node002    LINUX       INTEL  Unclaimed  Idle        0.000   503
0+00:00:15
vm2@node002    LINUX       INTEL  Unclaimed  Idle        0.000   503
0+00:00:16
vm1@node003    LINUX       INTEL  Unclaimed  Idle        0.000   503
0+00:00:15
vm2@node003    LINUX       INTEL  Unclaimed  Idle        0.000   503
0+00:00:16
      .
      .
      .

                      Machines Owner Claimed Unclaimed Matched Preempting

         INTEL/LINUX       20    19       0         1       0          0

               Total       20    19       0         1       0          0
```